



Performance Analysis of Momentum of Adam Optimizer on YOLO-V8 Using Traffic Object Dataset



Madhura M. Bhosale^{1*}, Yogesh S. Angal²

¹ Department of Electronics and Telecommunication Engineering, JSPM's Rajarshi Shahu College of Engineering, Pune 411033, India

² Department of Electronics and Telecommunication Engineering, JSPM's Bhivarabai Sawant Institute of Technology & Research, Pune 412207, India

Corresponding Author Email: Madhurabhosale4@gmail.com

Copyright: ©2025 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ts.420143>

ABSTRACT

Received: 22 March 2024

Revised: 29 August 2024

Accepted: 15 October 2024

Available online: 28 February 2025

Keywords:

object detection, machine learning, optimizer

In the past ten years, object detection has evolved significantly. Deep learning offers a big advantage for this task. The recent YOLO-V8 development makes object detection faster and more reliable. The optimizer is crucial in neural networks because it helps achieve accurate results. Choosing the right optimizer with the right parameters is very important. Currently, Adaptive Moment Estimation (Adam) works well among optimizers. Adam combines Momentum and RMSProp to efficiently adjust the weights of a neural network. In our work, we test the open-source KITTI dataset, looking at performance using metrics like Recall, Precision, F1 score, and mAP accuracy. We examine how different momentum values of the Adam optimizer affect traffic object detection. We choose momentum values of 0.222, 0.555, and 0.999. The 0.555 momentum with Adam optimizer performs the best, achieving mAP50 and mAP95 accuracies of 90% and 70%, respectively. We train 5,984 images using an NVIDIA RTX A5000 GPU. Training times for momentum values of 0.222, 0.555, and 0.999 are 3.482, 3.271, and 3.375 hours, respectively. Adam optimizer with 0.555 momentum reduces both training time and inference time on video data compared to the other values. Inference times are 8.4, 5.9, and 13.1 milliseconds for momentum values of 0.222, 0.555, and 0.999, respectively. The 0.555 momentum halves the prediction time compared to the other values.

1. INTRODUCTION

Object detection plays an important role in the field of autonomous vehicles. In the last ten years, the evolution in object detection is remarkable. Recent developments in deep learning provide a significant edge for object detection. This evolution starts with traditional object detection algorithms. The first major improvement in traditional object detection comes with the Viola-Jones detector in 2001. This algorithm is mainly used for real-time human face detection and operates efficiently on a 700MHz Pentium III CPU, making it ten times or more faster than other algorithms at that time [1]. Viola-Jones is a straightforward solution for detection using a sliding window approach, which scans all possible locations and scales in an image to detect any human face present. It improves accuracy using techniques such as integral images, feature selection, and detection cascades. Later, in 2005, N. Dalal and B. Triggs propose the Histogram of Oriented Gradients (HOG) [2]. The Histogram of Oriented Gradients is an important improvement for scale-invariant feature transformation. It is designed to balance feature invariance and non-linearity. In 2008, the deformable part-based model is introduced [3], following a detection and refinement approach. After 2012, the era of deep learning begins. Deep learning

algorithms fall into two categories: two-stage detection and one-stage detection. Object detection using selective search is introduced by Uijlings et al. [4] and Szegedy et al. [5] explains how deepening convolutional layers helps extract more meaningful features from images. Lowe [6] provide a method for object recognition using local invariant scale features. As handcrafted features of images reach their limit, deep learning evolves to offer new methods for calculating high-level features. Girshick [7] introduced R-CNN. In R-CNN, each image is rescaled to different levels, and these images are fed into a CNN to calculate features. These features are then used by a classifier to determine if an object is present in a specific area. Ren et al. [8] proposed Fast R-CNN, which improves upon R-CNN by training both bounding box regression and detection on the same network. This makes Fast R-CNN 200 times faster than R-CNN and includes a region proposal network. Faster R-CNN represents the first unified architecture, where each block is linked in a way that effectively represents object detection. The researchers [9-11] proposed You Only Look Once (YOLO), which is the first one-stage object detection algorithm in the era of neural networks. YOLO adopts a different paradigm from two-stage detection by dividing input images into grids. When the center of an object falls on a grid, that grid is responsible for detecting

the object [12]. Andrie Dazlee et al. [13] discuss how YOLO is useful for object detection in the field of autonomous vehicles. Liang et al. [14] provide insights into real-time object detection through cloud-based operations on autonomous vehicle data. Blaschko and Lampert [15] present results on localizing objects using structured output regression. Zou [16] explain the evolution of object detection. Terven et al. [17] provide a comparative study of the evolution from YOLOv1 to YOLOv8 and YOLO-NAS. Han et al. [18] discuss target fusion and camera-based object detection using an improved version of YOLO. YOLO creates a significant revolution in object detection. The optimizer plays an important role in neural network training for object detection. Finding the best set of parameters for a problem is the main task of an optimizer. Figure 1 shows the overall workflow of an optimizer in neural network training.

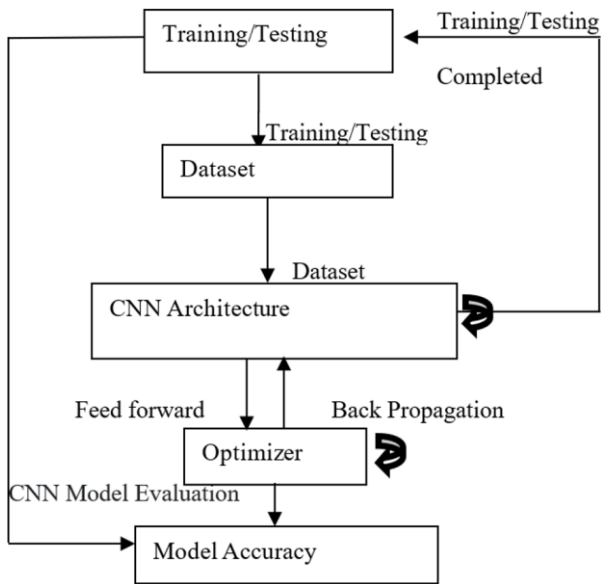


Figure 1. Sequence diagram of the optimization technique

Several optimization techniques, including SGD [19], Adagrad, and Adam, each come with their own set of advantages and disadvantages. Research indicates that the Adam optimizer often outperforms other methods. It shows strong generalization across different datasets and allows neural networks to converge more efficiently and accurately. A significant benefit of Adam is its capability to manage noisy and sparse datasets, as well as its effectiveness in handling a wide range of parameter values. In comparison to methods like SGD and RMSProp, Adam typically produces better outcomes. SGD is a simple optimization method, but it tends to converge slowly and can get stuck in local minima. A variant of SGD [20], known as Momentum, can also encounter issues such as getting trapped at saddle points or showing oscillatory behavior near the minimum. NAG, another optimization technique, uses a look-ahead strategy to predict the gradient of the cost function for the next step. However, it may face challenges with instability and oscillation. AdaGrad [21, 22] adjusts the learning rate dynamically, while AdaDelta [23, 24] uses exponential decay to control the learning rate, although it often moves slowly during the initial training stages. RMSProp modifies the learning rate for each parameter based on a moving average of squared gradients, but it can sometimes demand considerable memory and experience instability. Adam is recognized as a highly effective optimizer

for deep learning models [25]. By merging an adaptive learning rate with momentum-based updates, it enables quicker and more accurate convergence to an optimal set of parameters that minimize the cost or loss function.

In Figure 1, we observe that initially, we need to segregate data into training and testing parts. Next, we select an appropriate CNN architecture and optimization techniques to train our dataset. Backpropagation helps the algorithm iterate the neural network in the backward direction to learn more features from the dataset and update the weights. With each epoch, accuracy improves. In a deep neural network, we aim to find parameters that minimize the loss function, achieving the lowest loss function using those parameters. The goal is to reach a smaller function value at the global minimum compared to other points. Generally, the optimizer function is convex, where optimizers like SGD and RMSProp converge towards the global minimum. Figure 2 shows the graph of the cost function for every value of θ , illustrating how convergence occurs at local and global minimum points, as well as the location of the initial weight and how weights change with the gradient slope. Adam optimizer, which stands for Adaptive Moment Estimation, is an adaptive learning algorithm designed to improve training speed in deep neural networks and converge more easily.

Based on gradient history, Adam customizes adjustments to help the neural network learn quickly. Standard gradient descent provides the foundation, represented by equation 1, where θ is the model parameter, α is the learning rate, and gt is the gradient of the cost function with respect to the model parameters.

$$\theta = \theta - \alpha * gt \tag{1}$$

The parameter update, represented as θ , shifts in the direction of the negative gradient to minimize the cost function. The learning rate, α , determines the size of each step taken during the update. In standard gradient descent, α stays constant, which means that training starts with a predetermined learning rate. Modifications to α can be implemented in discrete increments or through various methods to enhance optimization efficiency.

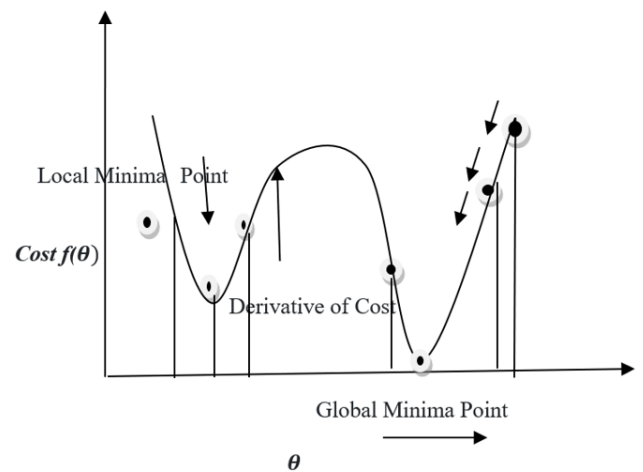


Figure 2. Local and global minimum point

If the learning rate is set too low, the convergence process can be sluggish, while a high learning rate might lead the model to overshoot the optimal minimum. The Adam optimizer addresses this issue by providing an adaptive

learning rate for each parameter. By modifying the learning rate according to the gradient history, Adam improves the efficiency of training neural networks. It combines aspects of Momentum and Root Mean Square Propagation (RMSProp). Recently, optimizers like AdamW and Ranger have emerged as competitors to Adam, but they require more computational resources and additional hyperparameter tuning. For this reason, we focus on Adam due to its lower complexity and easier hyperparameter tuning. In this work, we primarily evaluate the Adam optimizer with different momentum values on YOLO-V8 object detection.

1.1 Momentum role in Adam optimizer

Momentum improves training speed by amplifying gradients in the right direction. It achieves this by adding a portion of the previous gradient to the current one. When the gradient consistently moves in the same direction across several iterations, the accumulated momentum term—related to past gradients—facilitates a faster optimization process. Gradient descent generally functions like rolling a ball down a hill, taking fixed steps because the learning rate remains constant throughout. This means the gradient is calculated each step in the direction of alpha. Momentum techniques help recognize that if the last few steps have been in the same direction, fewer steps are needed.

$$vt = \gamma * vt - 1 + n * gt \tag{2}$$

$$\theta = \theta - vt \tag{3}$$

From Eqs. (2) and (3), it is clear that subtracting the momentum and updating the parameter θ differs from the gradient descent algorithm. At time t the momentum vector vt is influenced by the previous momentum vector $vt-1$. The hyperparameter γ controls momentum decay, applying an exponential reduction to past momentum values. The learning rate η determines the step size in the direction opposite to the gradient.

2. METHODOLOGY

In the YOLO-V8 framework, a momentum value of 0.999 is used for the Adam optimizer. To evaluate this choice, we need to compare it with other momentum values. The range for the momentum value in the Adam optimizer is between 0 and 1. Given that 0.999 is already in use, we will compare it with lower and medium values. We select 0.222 as the lower value and 0.555 as the medium value, in addition to the default value of 0.999. The methodology is divided into two parts: training the object detection model and testing the object detection model.

Figure 3(a) shows the block diagram of the object detection training model. First, we divide the input data into training and validation sets, and then provide this data to the object detection machine learning model. For our experiments, we use YOLO-V8, which is a recent development in the field of object detection.

Hyperparameters selection plays crucial role in machine learning model training. For this work we have used Batch size = 16, Image size = 640, Epochs = 150, Yolo variant = Yolov8-x, Optimizer = Adam with momentum values 0.222, 0.555, 0.999 respectively.

Once model trained on training data and validate on validation data then we need to provide unseen data to check the predictions. Figure 3(b) represents testing block diagram of object detection. Figure 4 gives idea about flowchart of workflow.

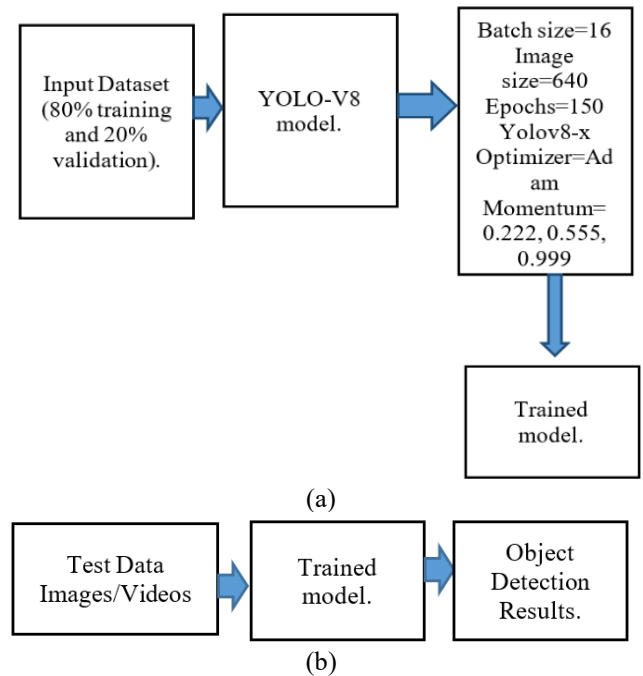


Figure 3. (a) Block diagram of object detection model training; (b) Block diagram of object detection model testing

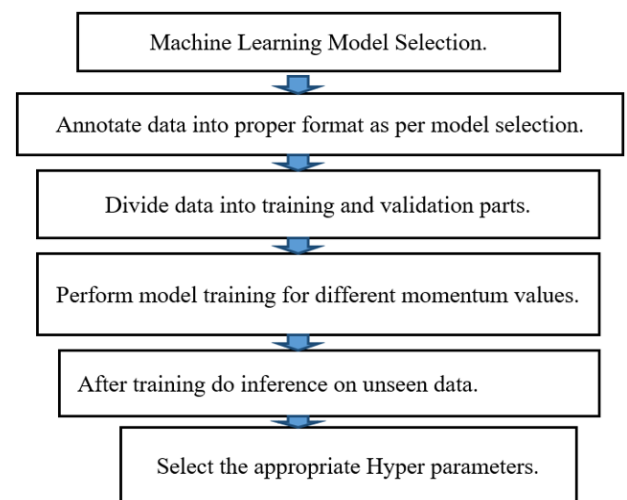


Figure 4. Flowchart of system

3. RESULTS

The experiment described in this work is conducted on Python version 3.9.13 with Windows 10 Pro OS, 32 GB RAM, and a 2nd Gen Intel(R) Core (TM) i9-12900 2.40 GHz processor. We use an Nvidia RTX A5000 GPU with 12 GB RAM, along with torch-2.0.1 and CUDA version 11.7. For this study, we utilize the KITTI [26] open-source dataset, which includes classes such as car, pedestrian, van, cyclist, truck, misc, tram, and person sitting. We use approximately 5,984 images for training and 1,400 images for validation.

The performance of YOLO-V8 is evaluated using different

momentum values with the Adam optimizer. We adjust the Adam optimizer's momentum to 0.002, 0.005, and 0.999 and assess the performance of the object detection model. We conduct experiments with these momentum values and set hyper parameters to epochs 150, batch size 16, and image size 640, using the YOLOv8-x variant. we observe that the mAP 50 and 95 accuracies for a momentum of 0.222 are less than 90% and less than 70%, respectively. For a momentum of 0.555, the mAP 50 and 95 accuracies exceed 90% and 70%. For a momentum of 0.999, the mAP 50 and 95 accuracies are less than 80% and less than 60%, respectively.

3.1 Object detection on image data

Figure 5 illustrates the confusion matrix with the 0.222 momentum value, and Figure 6 depicts the F1-confidence curve achieved with the same momentum. Figures 6 and 7 show the F1 curve and precision-confidence for each class with a momentum value of 0.222, respectively. Table 1 presents the results on the validation dataset using the 0.222 momentum value.

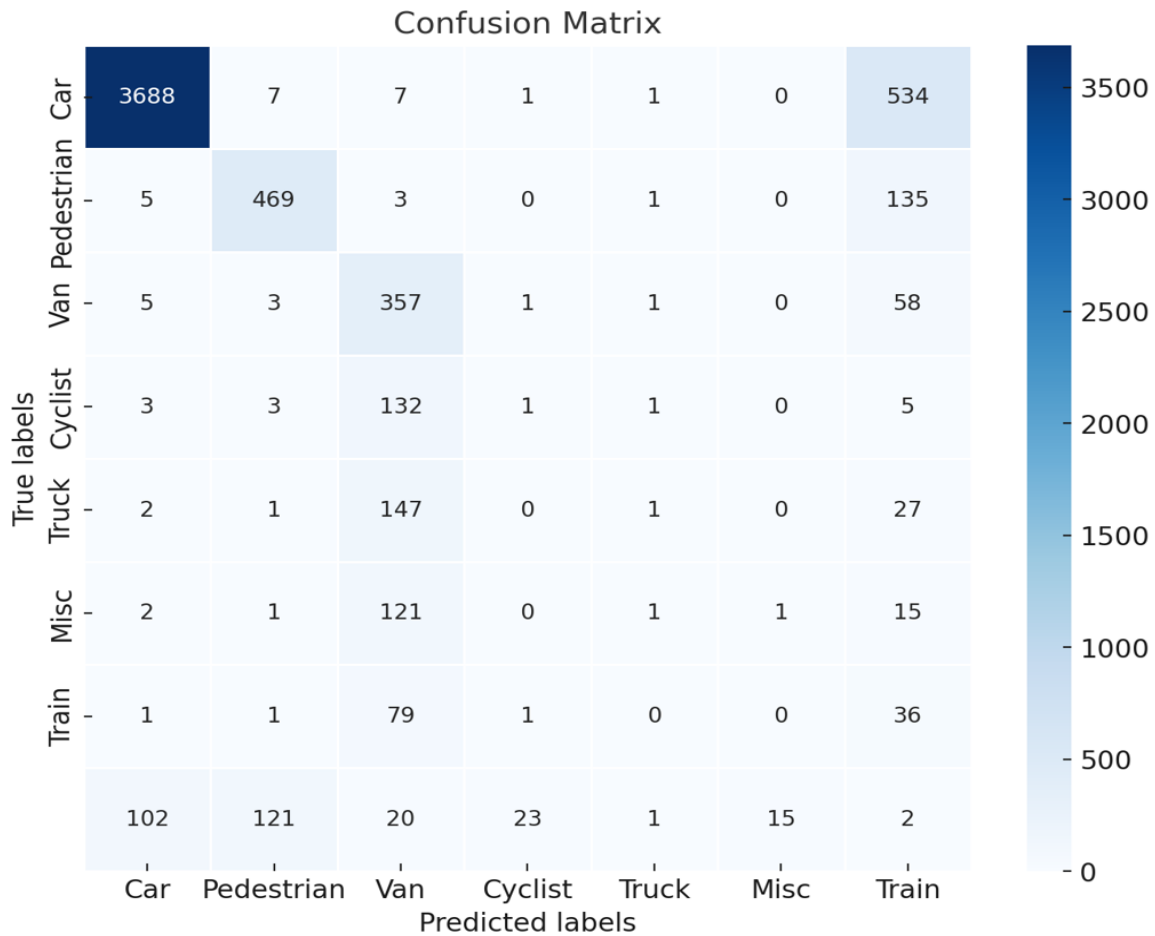


Figure 5. Confusion matrix with 0.222 momentum

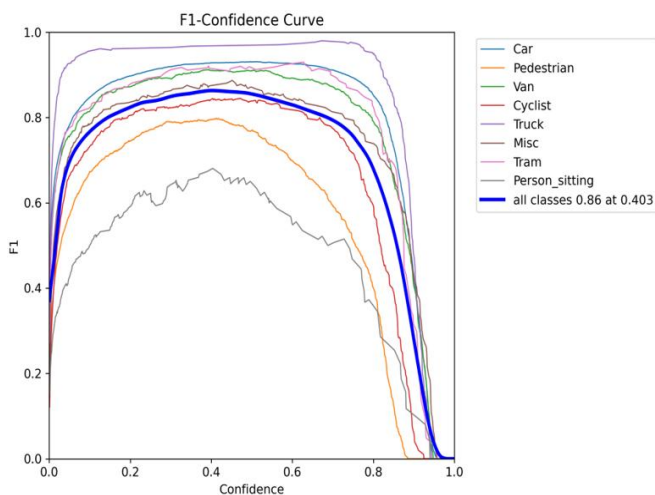


Figure 6. F1 curve with 0.222 momentum

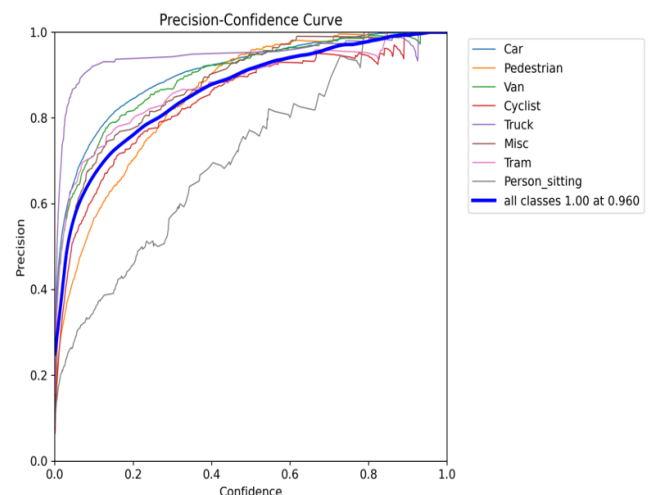


Figure 7. Precision-confidence with 0.222 momentum

Table 1. Validation results with Adam optimizer for 0.222 momentum

Class	Images	Instances	Box (P)	Box (R)	mAP (50)	mAP (95)
All	1045	5474	0.885	0.854	0.899	0.667
Car	1045	3888	0.927	0.929	0.97	0.812
Pedestrian	1045	585	0.921	0.699	0.822	0.489
Van	1045	386	0.925	0.894	0.948	0.764
Cyclist	1045	198	0.875	0.81	0.87	0.608
Truck	1045	150	0.951	0.987	0.984	0.837
Misc	1045	139	0.91	0.849	0.918	0.683
Tram	1045	80	0.883	0.943	0.958	0.714
Person Sitting	1045	48	0.69	0.648	0.72	0.427

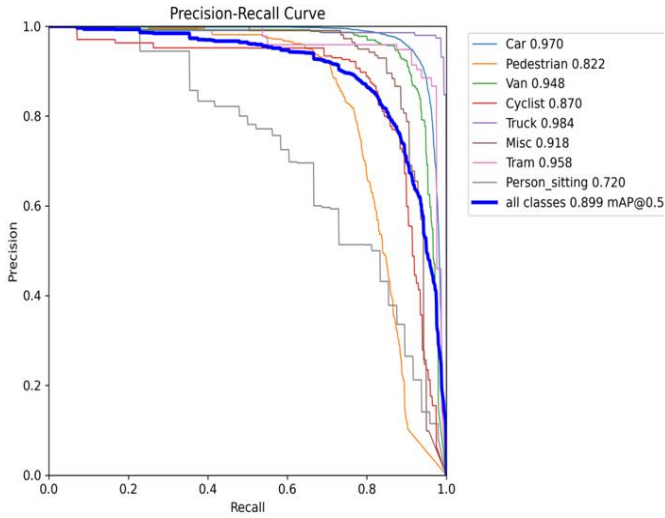


Figure 8. Precision-recall with 0.222 momentum

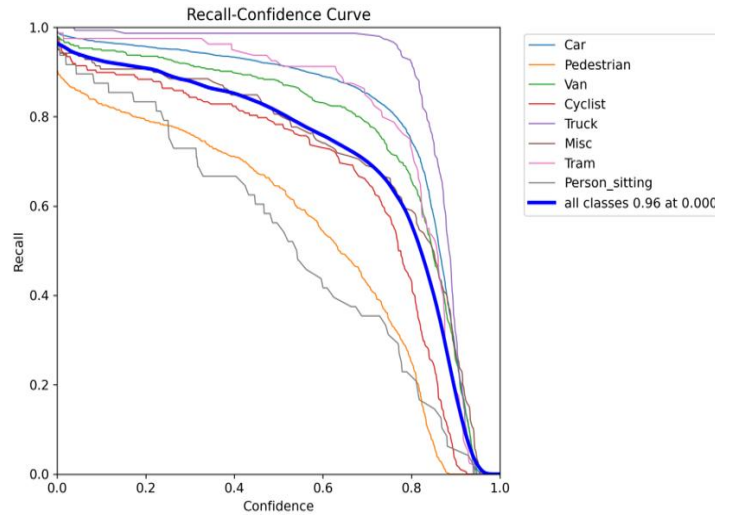


Figure 9. Recall-confidence with 0.222 momentum

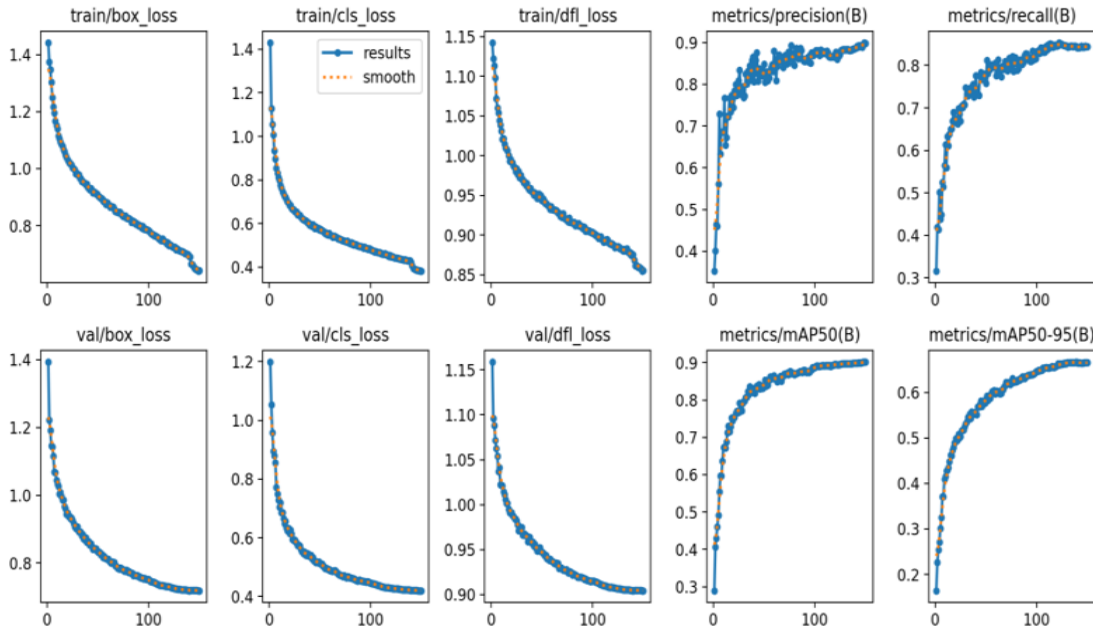


Figure 10. Performance analysis with 0.222 momentum

Figures 8-10 display the Precision-recall curve, Recall-confidence, and Performance analysis curves, respectively. In the performance analysis curve, we observe the mAP50 and mAP95 accuracy percentages for the 0.222 momentum value. The mAP50 value is less than 90%, while the mAP95 value is 66%.

Figure 11 shows a training sample image, while Figure 12 illustrates the training time required with a 0.222 momentum

value.

Figures 13-16 display the confusion matrix, performance analysis graphs, F1-confidence curve, and Precision-confidence curve, respectively, for a 0.555 momentum value. Table 2 presents the validation results using the Adam optimizer with a 0.555 momentum value. In the performance analysis curve, we see that the mAP50 value is 90% and the mAP95 value is 70% for the 0.555 momentum.



Figure 11. Training images with 0.222 momentum

150 epochs completed in 3.482 hours.
 Optimizer stripped from runs\detect\train8\weights\last.pt, 6.2MB
 Optimizer stripped from runs\detect\train8\weights\best.pt, 6.2MB

Validating runs\detect\train8\weights\best.pt...
 Ultralytics YOLOv8.0.229 Python-3.9.13 torch-2.0.1+cu117 CUDA:0 (NVIDIA RTX A5000, 24564MiB)
 Model summary (fused): 168 layers, 3007208 parameters, 0 gradients, 8.1 GFLOPs

Figure 12. Training time with 0.222 momentum

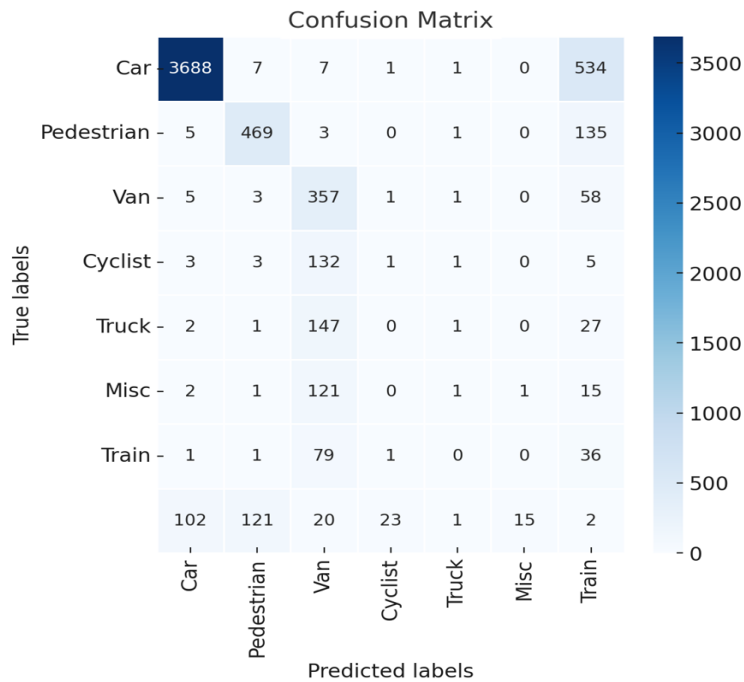


Figure 13. Confusion matrix with 0.555 momentum

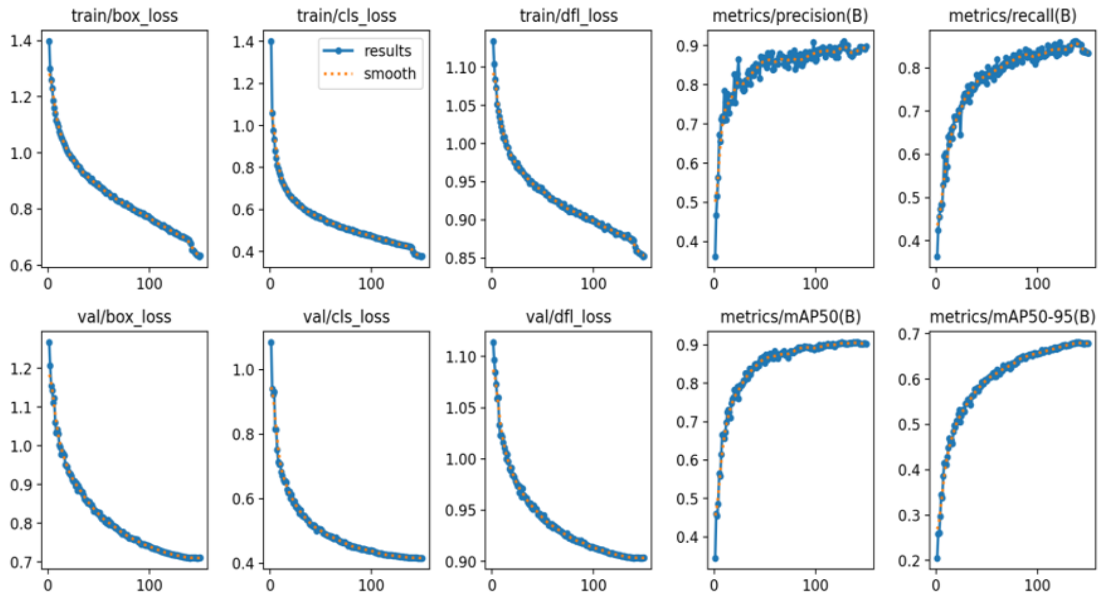


Figure 14. Performance analysis with 0.555 momentum

Table 2. Validation results with Adam optimizer for 0.555 momentum

Class	Images	Instances	Box (P)	Box (R)	mAP (50)	mAP (95)
All	1045	5474	0.883	0.858	0.906	0.682
Car	1045	3888	0.927	0.933	0.971	0.815
Pedestrian	1045	585	0.899	0.679	0.811	0.467
Van	1045	386	0.918	0.909	0.947	0.775
Cyclist	1045	198	0.828	0.833	0.878	0.614
Truck	1045	150	0.949	1	0.985	0.85
Misc	1045	139	0.926	0.878	0.921	0.694
Tram	1045	80	0.949	0.924	0.973	0.759
Person Sitting	1045	48	0.669	0.708	0.766	0.479

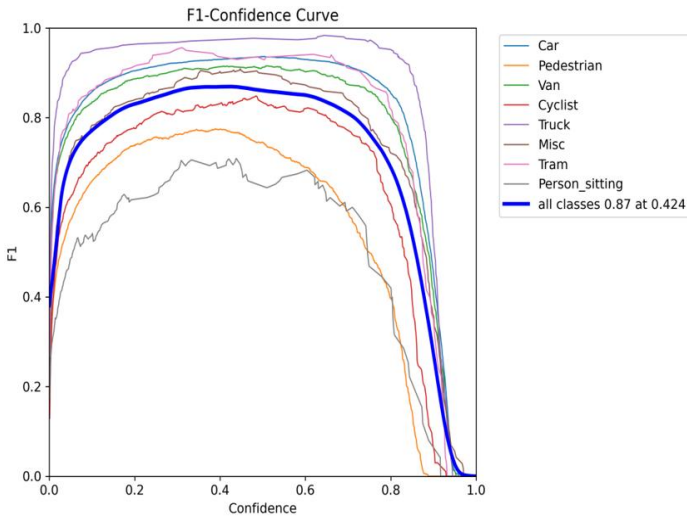


Figure 15. F1-confidence curve with 0.555 momentum

Figures 17 and 18 represent the Precision-Recall and Recall-Confidence curves, respectively, for a momentum value of 0.555. Figure 19 shows a sample training image, and Figure 20 indicates the training time required to train the data with a momentum value of 0.555.

Figures 21-23, along with Table 3, represent the confusion matrix, F1-Confidence curve, and Precision-Confidence curve, respectively, for a momentum value of 0.999. These figures and table display the validation results for each class

using this momentum value.

Figures 24-28 represent the Precision-Recall Curve, Recall-Confidence curve, Performance Analysis, Training Time, and a Training Sample Image, respectively, for a momentum value of 0.999. In the Performance Analysis curve, we can see that the mAP50 and mAP95 accuracy percentage values for the 0.999 momentum are shown. The mAP50 value is less than 80%, and the mAP95 value is around 60%.

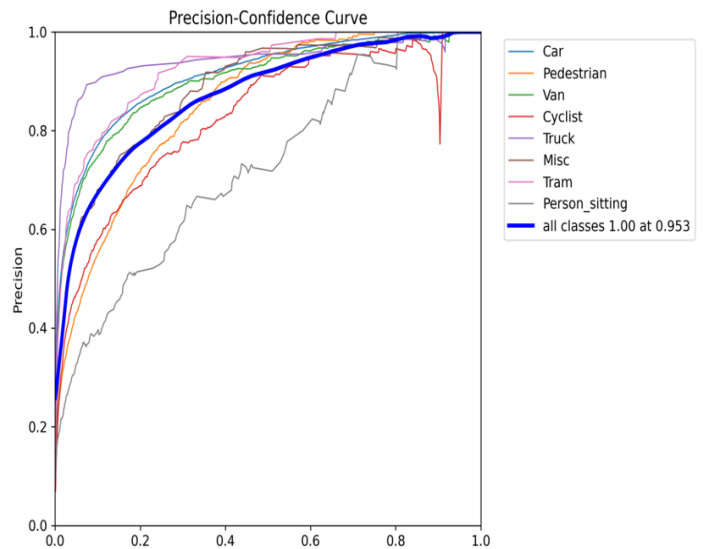


Figure 16. Precision-confidence with 0.555 momentum

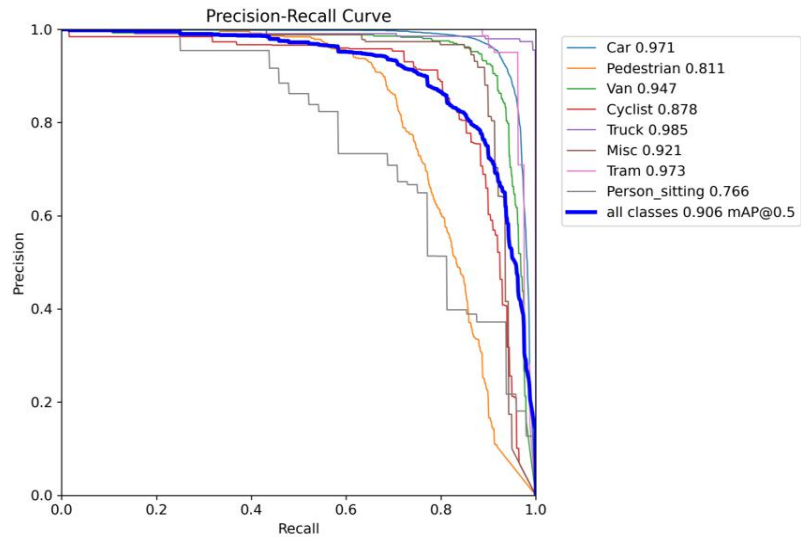


Figure 17. Precision-recall with 0.555 momentum

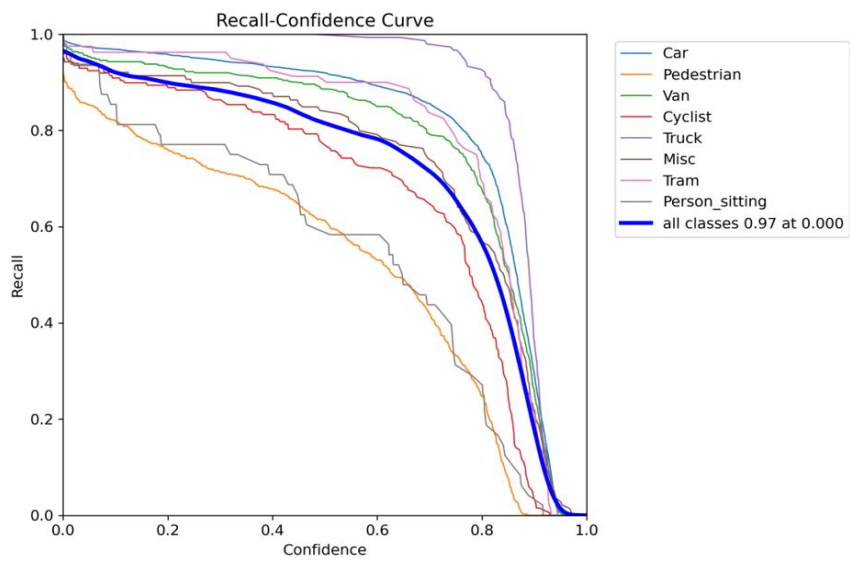


Figure 18. Recall-confidence with 0.555 momentum

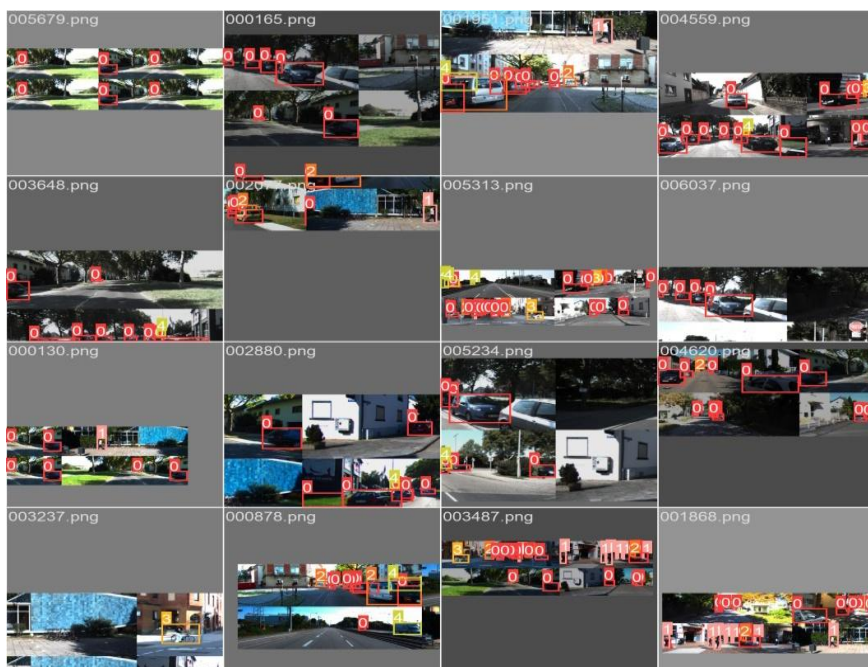


Figure 19. Training images with 0.555 momentum

150 epochs completed in 3.271 hours.

Optimizer stripped from runs\detect\train9\weights\last.pt, 6.2MB

Optimizer stripped from runs\detect\train9\weights\best.pt, 6.2MB

Validating runs\detect\train9\weights\best.pt...

Ultralytics YOLOv8.0.229 🚀 Python-3.9.13 torch-2.0.1+cu117 CUDA:0 (NVIDIA RTX A5000, 24564MiB)

Model summary (fused): 168 layers, 3007208 parameters, 0 gradients, 8.1 GFLOPs

Figure 20. Training time with 0.555 momentum

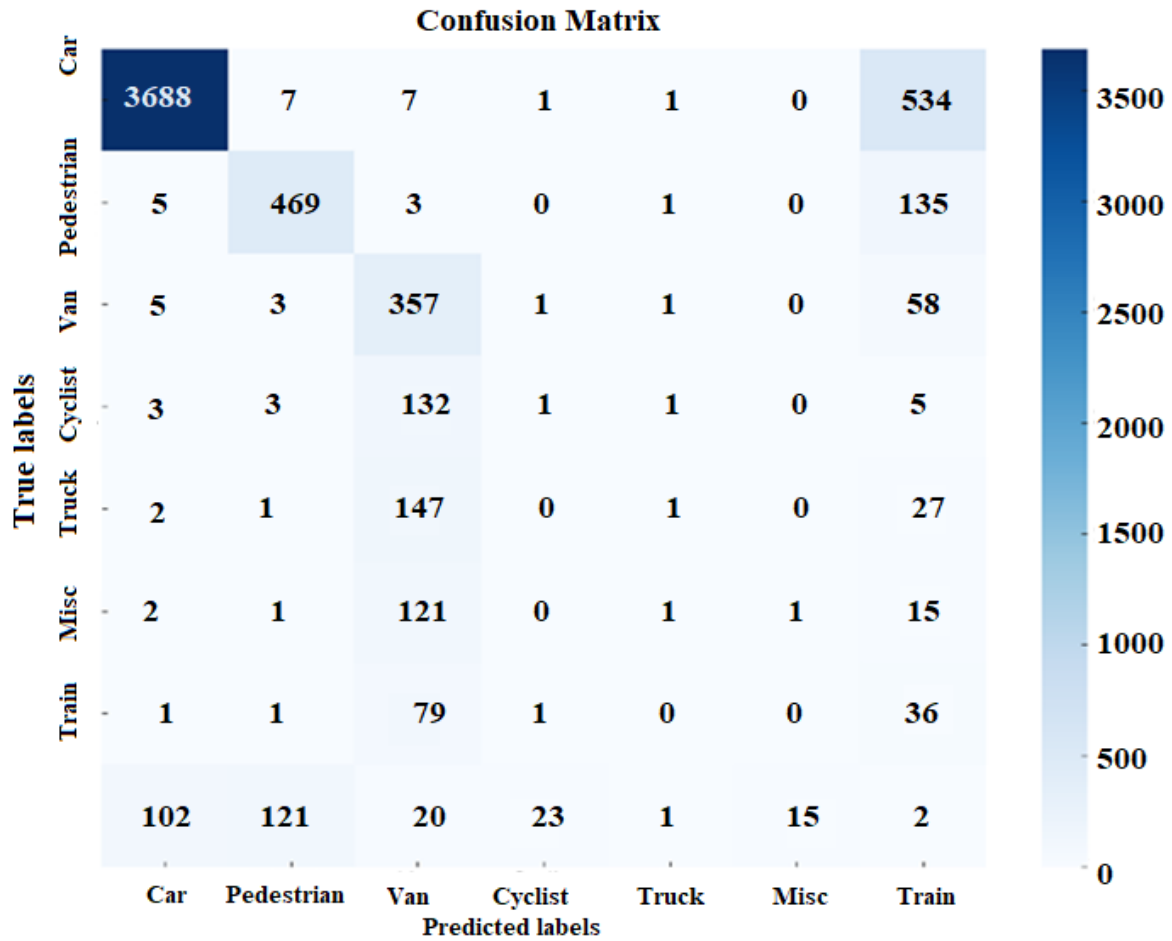


Figure 21. Confusion matrix with 0.999 momentum

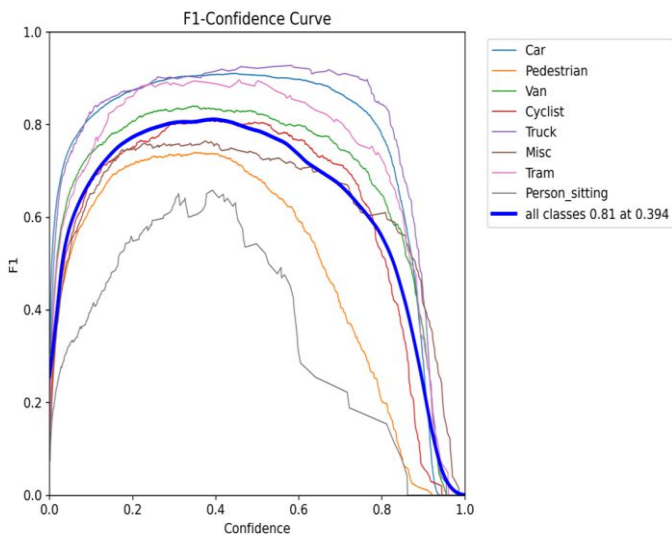


Figure 22. F1-confidence curve with 0.999 momentum

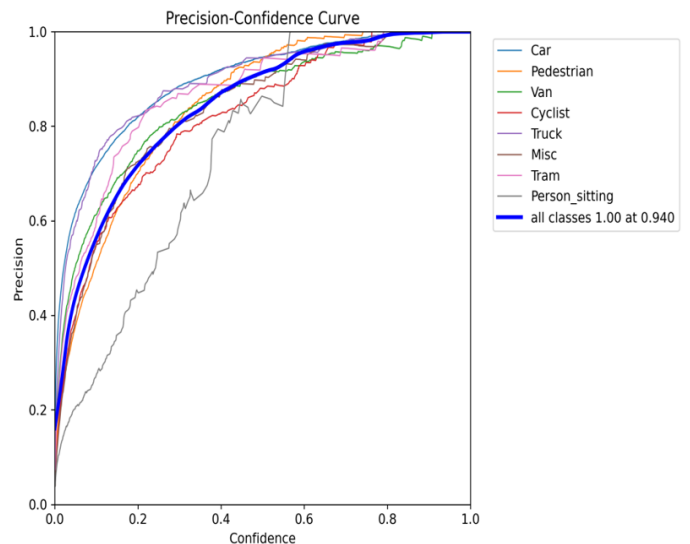


Figure 23. Precision-confidence with 0.999 momentum

Table 3. Validation result with Adam optimizer for 0.999 momentum

Class	Images	Instances	Box (P)	Box (R)	mAP (50)	mAP (95)
All	1045	5474	0.883	0.858	0.906	0.682
Car	1045	3888	0.927	0.933	0.971	0.815
Pedestrian	1045	585	0.899	0.679	0.811	0.467
Van	1045	386	0.918	0.909	0.947	0.775
Cyclist	1045	198	0.828	0.833	0.878	0.614
Truck	1045	150	0.949	1	0.985	0.85
Misc	1045	139	0.926	0.878	0.921	0.694
Tram	1045	80	0.949	0.924	0.973	0.759
Person Sitting	1045	48	0.669	0.708	0.766	0.479

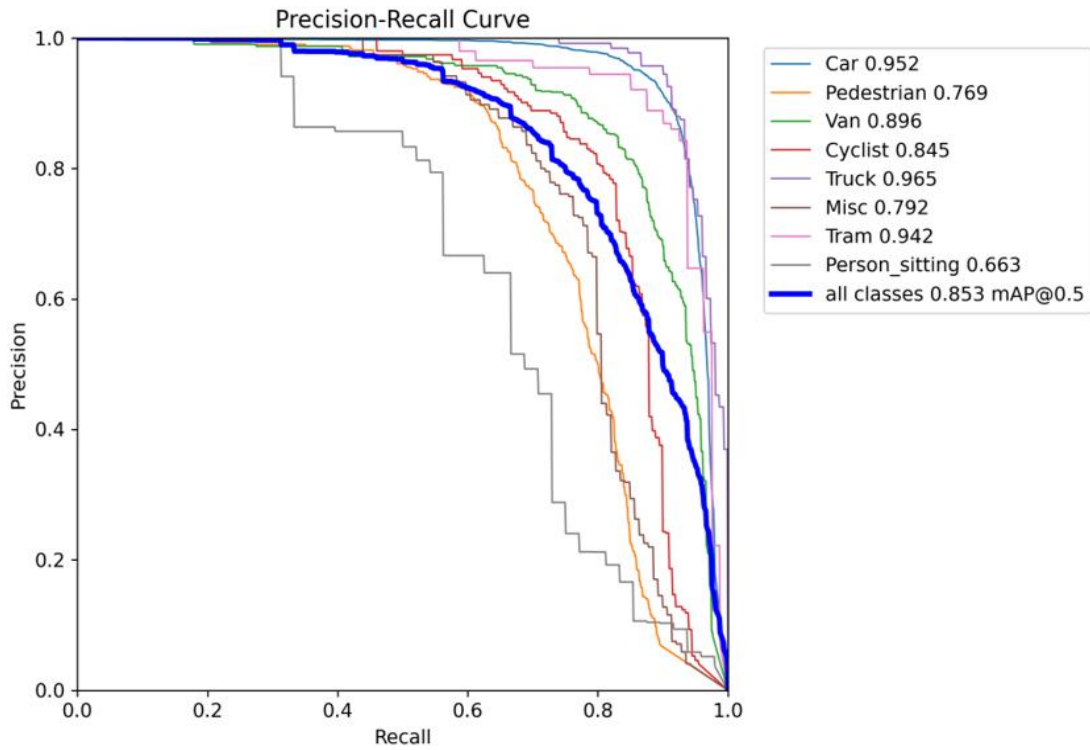


Figure 24. Precision-recall with 0.999 momentum

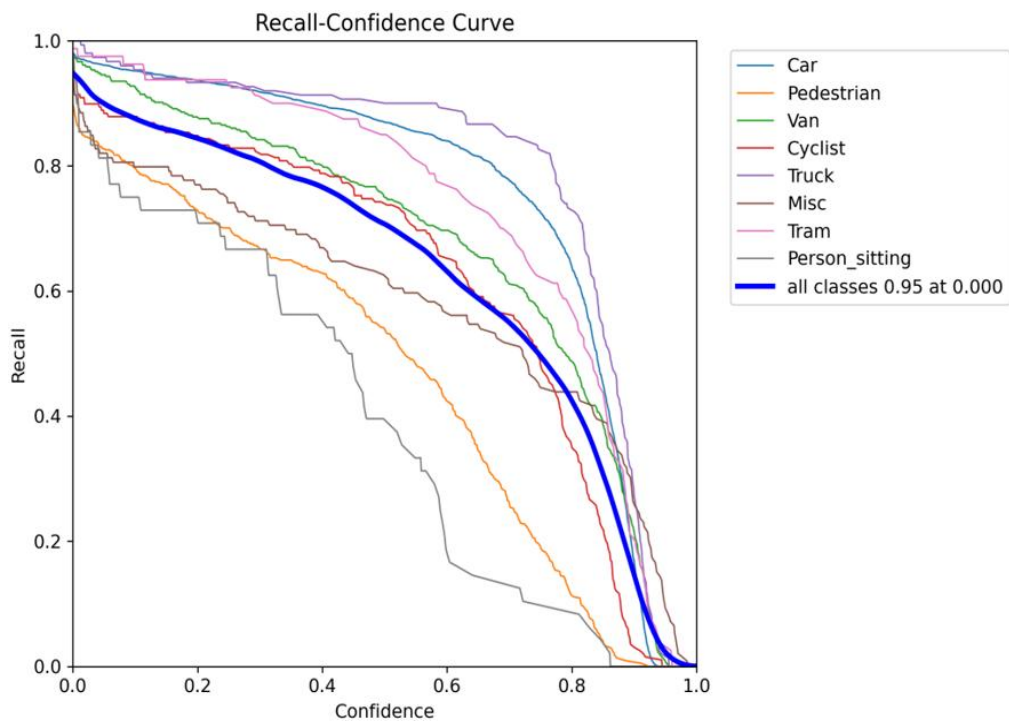


Figure 25. Recall-Confidence with 0.999 momentum

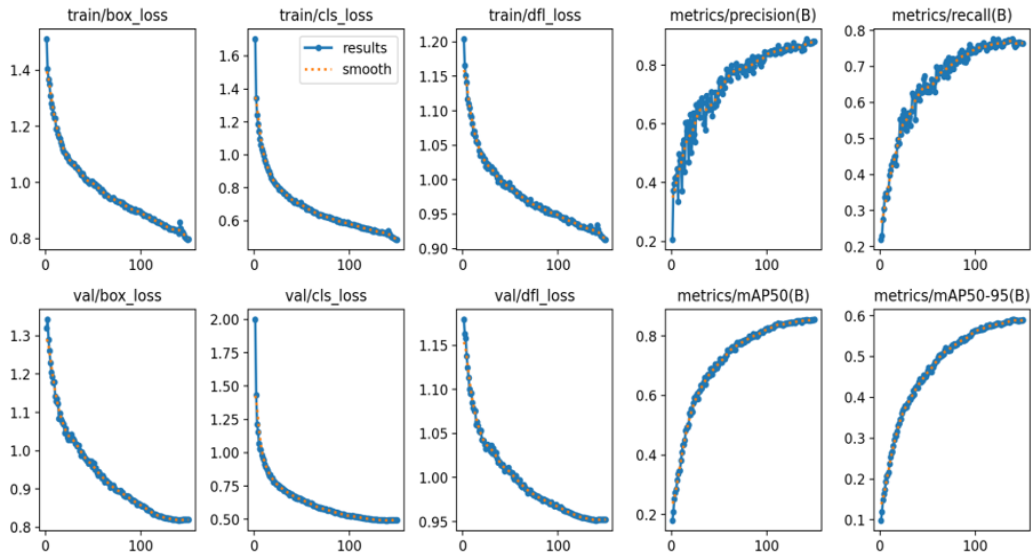


Figure 26. Performance analysis with 0.999 momentum

150 epochs completed in 3.375 hours.

Optimizer stripped from runs\detect\train10\weights\last.pt, 6.2MB

Optimizer stripped from runs\detect\train10\weights\best.pt, 6.2MB

Validating runs\detect\train10\weights\best.pt...

Ultralytics YOLOv8.0.229 🚀 Python-3.9.13 torch-2.0.1+cu117 CUDA:0 (NVIDIA RTX A5000, 24564MiB)

Model summary (fused): 168 layers, 3007208 parameters, 0 gradients, 8.1 GFLOPs

Figure 27. Training time with 0.999 momentum



Figure 28. Training images with 0.999 momentum

3.2 Object detection on video data

We apply the object detection trained model with momentum values of 0.222, 0.555, and 0.999 on a video. For comparative analysis, we use the KITTI video dataset, which has 85 frames. Using the Adam optimizer, the detection times are approximately 8.4 ms, 5.9 ms, and 13.1 ms for momentum values of 0.222, 0.555, and 0.999, respectively. You can see the results in Figures 29-31. If you use the Adam optimizer with the proper momentum, the inference time will reduce.

Table 4 explains about comparative analysis between different momentum values of Adam optimizer. We have p-value test to evaluate the statistical significance of accuracy and training time. For momentum values of 0.222, 0.555, and 0.999, the p-values are 1.13, 4.81, and 2.71, respectively, with a hypothesized value of 90. A lower p-value indicates that the result is more divergent from the hypothesis. From these results, we see that the momentum value of 0.555 is less divergent from the hypothesis compared to the other two momentum values for the Adam optimizer.

Table 4. Comparative analysis between different momentum values of Adam optimizer

No.	YOLO Version	YOLO Variant	Momentum Value	mAP50 (%)	mAP95 (%)	Training Time (hours)	Inference Time on Video
1	YOLO-V8	YOLOV8-x	0.222	90	66	3.482	8.4 ms
2	YOLO-V8	YOLOV8-x	0.555	90	70	3.271	5.9 ms
3	YOLO-V8	YOLOV8-x	0.999	80	60	3.375	13.1 ms

```
video 1/1 (70/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Misc, 7.0ms
video 1/1 (71/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 1 Van, 7.0ms
video 1/1 (72/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 8.0ms
video 1/1 (73/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 7.0ms
video 1/1 (74/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 7.0ms
video 1/1 (75/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 7.0ms
video 1/1 (76/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 7.0ms
video 1/1 (77/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 7.0ms
video 1/1 (78/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 7.0ms
video 1/1 (79/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 1 Van, 8.0ms
video 1/1 (80/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 7.0ms
video 1/1 (81/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 3.1ms
video 1/1 (82/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 7.0ms
video 1/1 (83/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 8.0ms
video 1/1 (84/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 8 Cars, 7.0ms
video 1/1 (85/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 8 Cars, 7.0ms
Speed: 1.2ms preprocess, 8.4ms inference, 1.7ms postprocess per image at shape (1, 3, 224, 640)
```

Figure 29. Inference time on video 0.222 momentum

```
video 1/1 (70/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 3 Cars, 0.0ms
video 1/1 (71/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 4 Cars, 0.0ms
video 1/1 (72/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 4 Cars, 0.0ms
video 1/1 (73/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 4 Cars, 0.0ms
video 1/1 (74/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 4 Cars, 0.0ms
video 1/1 (75/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 0.0ms
video 1/1 (76/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 8.0ms
video 1/1 (77/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 8.0ms
video 1/1 (78/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 6.1ms
video 1/1 (79/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 0.0ms
video 1/1 (80/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 0.0ms
video 1/1 (81/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 0.0ms
video 1/1 (82/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 0.0ms
video 1/1 (83/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 9.1ms
video 1/1 (84/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 6.0ms
video 1/1 (85/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 5.0ms
Speed: 1.2ms preprocess, 5.9ms inference, 1.2ms postprocess per image at shape (1, 3, 224, 640)
```

Figure 30. Inference time on video 0.555 momentum

```
video 1/1 (70/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 22.1ms
video 1/1 (71/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 5 Cars, 65.8ms
video 1/1 (72/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 5 Cars, 3.1ms
video 1/1 (73/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 6.0ms
video 1/1 (74/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 6.0ms
video 1/1 (75/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 6 Cars, 1 Van, 6.0ms
video 1/1 (76/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 4.0ms
video 1/1 (77/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 9.5ms
video 1/1 (78/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 0.0ms
video 1/1 (79/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 0.0ms
video 1/1 (80/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 7.0ms
video 1/1 (81/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 6.0ms
video 1/1 (82/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 6.0ms
video 1/1 (83/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 7 Cars, 6.0ms
video 1/1 (84/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 8 Cars, 6.0ms
video 1/1 (85/85) C:\Users\MQSTechnologies\Downloads\0027.mp4: 224x640 8 Cars, 0.0ms
Speed: 2.6ms preprocess, 13.1ms inference, 2.4ms postprocess per image at shape (1, 3, 224, 640)
```

Figure 31. Inference time on video 0.999 momentum

4. CONCLUSIONS

In this paper, we use the YOLO-V8 deep learning model with the Adam optimizer and perform experiments on different momentum values of the optimizer. We choose a default momentum value of 0.999 and compare it with two other values: 0.222 and 0.555. Our results show that, for the KITTI dataset, the Adam optimizer with a momentum value of 0.555 performs best compared to the other two values, achieving mAP50 and mAP95 accuracies of 90% and 70%, respectively. The training time also decreases with the proper momentum value selection; training the KITTI dataset with a momentum of 0.555 takes 3.271 hours, which is 20 minutes less than with a momentum of 0.222 and 10 minutes less than with a momentum of 0.999. For video inference, the momentum of 0.555 performs well, with an inference time of 5.9 ms, which is reduced by half compared to the 0.222 and 0.999 momentum values. Thus, using an appropriate optimization technique with the right momentum values helps the model converge in a shorter time, reducing both training and inference times. Tao et al. [26] published results on the KITTI dataset using a modified version of YOLO, OYOLO, and achieved a mAP-50 of around 80.1%. With our methodology on the same dataset, we achieve a mAP accuracy of around 90%, representing an improvement of almost 10% in accuracy.

In future scope focus needs to be given on to increase the dataset or to try with other datasets. You can improvise the results using other optimizers or with the help of modified version of optimizers.

ACKNOWLEDGMENT

We would like to show our gratitude to RSCOE Pune for sharing their pearls of wisdom with us during the course of the proposed work.

REFERENCES

- [1] Viola, P., Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, Kauai, HI, USA, pp. 1-11. <https://doi.org/10.1109/CVPR.2001.990517>
- [2] Dalal, N., Triggs, B. (2005). Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, USA, pp. 886-893. <https://doi.org/10.1109/CVPR.2005.177>
- [3] Felzenszwalb, P., McAllester, D., Ramanan, D. (2008). A discriminatively trained, multiscale, deformable part model. In 2008 IEEE Conference on Computer Vision and Pattern Recognition, Anchorage, AK, USA, pp. 1-8. <https://doi.org/10.1109/CVPR.2008.4587597>
- [4] Uijlings, J.R., Van De Sande, K.E., Gevers, T., Smeulders, A.W. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104: 154-171. <https://doi.org/10.1007/s11263-013-0620-5>
- [5] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). Going deeper with convolutions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, pp. 1-9. <https://doi.org/10.1109/CVPR.2015.7298594>
- [6] Lowe, D.G. (1999). Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision, Kerkyra, Greece, pp. 1150-1157. <https://doi.org/10.1109/ICCV.1999.790410>
- [7] Girshick, R. (2015). Fast R-CNN. In 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, pp. 1440-1448. <https://doi.org/10.1109/ICCV.2015.169>
- [8] Ren, S., He, K., Girshick, R., Sun, J. (2016). Faster R-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6): 1137-1149. <https://doi.org/10.1109/TPAMI.2016.2577031>
- [9] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 779-788. <https://doi.org/10.1109/CVPR.2016.91>
- [10] Dai, P., Yao, S., Li, Z., Zhang, S., Cao, X. (2022). ACE: Anchor-free corner evolution for real-time arbitrarily-oriented object detection. *IEEE Transactions on Image Processing*, 31: 4076-4089. <https://doi.org/10.1109/TIP.2022.3167919>
- [11] Amirkhani, A., Karimi, M.P., Banitalebi-Dehkordi, A. (2023). A survey on adversarial attacks and defenses for object detection and their applications in autonomous vehicles. *The Visual Computer*, 39(11): 5293-5307. <https://doi.org/10.1007/s00371-022-02660-6>
- [12] He, Q., Xu, A., Ye, Z., Zhou, W., Cai, T. (2023). Object detection based on lightweight YOLOX for autonomous driving. *Sensors*, 23(17): 7596. <https://doi.org/10.3390/s23177596>
- [13] Andrie Dazlee, N.M.A., Abdul Khalil, S., Abdul-Rahman, S., Mutalib, S. (2022). Object detection for autonomous vehicles with sensor-based technology using yolo. *International Journal of Intelligent Systems and Applications in Engineering*, 10(1): 129-134. <https://doi.org/10.18201/ijisae.2022.276>
- [14] Liang, S., Wu, H., Zhen, L., Hua, Q., Garg, S., Kaddoum, G. (2022). Edge YOLO: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(12): 25345-25360. <https://doi.org/10.1109/TITS.2022.3158253>
- [15] Blaschko, M.B., Lampert, C.H. (2008). Learning to localize objects with structured output regression. In Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, pp. 2-15. https://doi.org/10.1007/978-3-540-88682-2_2
- [16] Zou, X. (2019). A review of object detection techniques. In 2019 International Conference on Smart Grid and Electrical Automation (ICSGEA), Xiangtan, China, pp. 251-254. <https://doi.org/10.1109/ICSGEA.2019.00065>
- [17] Terven, J., Córdova-Esparza, D.M., Romero-González, J.A. (2023). A comprehensive review of yolo architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, 5(4): 1680-1716. <https://doi.org/10.3390/make5040083>

- [18] Han, J., Liao, Y., Zhang, J., Wang, S., Li, S. (2018). Target fusion detection of LiDAR and camera based on the improved YOLO algorithm. *Mathematics*, 6(10): 213. <https://doi.org/10.3390/math6100213>
- [19] Darken, C., Chang, J., Moody, J. (1992). Learning rate schedules for faster stochastic gradient search. In *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, Helsingoer, Denmark, pp. 3-12. <https://doi.org/10.1109/NNSP.1992.253713>
- [20] Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. <https://doi.org/10.48550/arXiv.1609.04747>
- [21] Uijlings, J.R., Van De Sande, K.E., Gevers, T., Smeulders, A.W. (2013). Selective search for object recognition. *International Journal of Computer Vision*, 104: 154-171. <https://doi.org/10.1007/s11263-013-0620-5>
- [22] Zeiler, M.D. (2012). ADADELTA: An adaptive learning rate method. arXiv preprint arXiv:1212.5701. <https://doi.org/10.48550/arXiv.1212.5701>
- [23] Cai, Z., Vasconcelos, N. (2018). Cascade R-CNN: Delving into high quality object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, pp. 6154-6162. <https://doi.org/10.1109/CVPR.2018.00644>
- [24] Dozat, T. (2016). Incorporating Nesterov momentum into Adam. Workshop Track - ICLR 2016. <https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ>
- [25] Kitti Dataset. <https://www.kaggle.com/datasets/klemenko/kitti-dataset>
- [26] Tao, J., Wang, H., Zhang, X., Li, X., Yang, H. (2017). An object detection system based on YOLO in traffic scene. In *2017 6th International Conference on Computer Science and Network Technology (ICCSNT)*, Dalian, China, pp. 315-319. <https://doi.org/10.1109/ICCSNT.2017.8343709>